WEST

Generate Collection | Print

Appeal conference — John Chavis

L2: Entry 41 of 42 File: USPT Jan 18, 2000

9/15/03

DOCUMENT-IDENTIFIER: US 6016477 A
TITLE: Method and apparatus for identifying applicable business rules

Abstract Text (1):
A method and apparatus for identifying decision points in Business Objects and classifying business rules that are applicable to the decision points. Business Objects are created which are decorated with business rules, either manually or programmatically. Control Points are used to represent the named decision points or triggers within the behavior of the Business Objects. The Control Points are visually exposed to business analysts to allow the examination of the business rules attached to the various Control Points. The business analysts are permitted to manually attach or detach business rules associated with the Control Points. A system can programmatically decide which rules to attach to a Control Point based on execution context.

Brief Summary Text (2):
The present invention relates to data processing systems, and more particularly, to externalizing business decisions into business rules in object-oriented objects which are describable and manipulatable by non-programmers.

Brief Summary Text (5):
Application Ser. No. 08/989,674 filed by David Ehnebuske and Barbara McKee entitled, "Method and Apparatus For Annotating Static Object Models With Business Rules" (IBM Docket AT9-97-499).

Brief Summary Text (10):
Implementing business rules within Business Objects enables businesses to automate their policies and practices. For example, business rules can control the flow through the tasks of a business process. The next task is performed only when the rules that permit ending the previous task and those that determine that the next task should be entered have been satisfied. Business rules can also assist in decision-making and have a significant impact on decision support applications. For example, a rule system can determine whether or not to extend credit to a customer and how much credit to extend.

Brief Summary Text (11):
Historically, business rules have been embedded within the logic of applications. These applications have been designed to perform business functions, horizontal applications such as accounts receivable and inventory control or vertical applications such as portfolio analysis in financial services and insurance eligibility in healthcare. Developers have built these systems without explicit regard for business rules. As a result, when business policies and practices change--and they're constantly changing--it's difficult to reflect those changes in the applications that implement them.

Brief Summary Text (12):
More recently, business rules have been implemented in database triggers. In response to database changes, database triggers are automatically invoked by a database server. The code in the triggers could execute some procedural logic as well as manipulate the database. Database triggers and stored procedures offer the advantage of modularity. They isolate business rules and technical data-manipulation rules from application logic. Triggers automate business rules processing and provide application independence (any application changing the database causes the triggers to be fired). However, triggers also have some serious disadvantages. They are hard to develop. They are intended to implement technical data-manipulation

rules as well as business rules, and they are hard to maintain and extend.

Brief Summary Text (14):
Database triggers function on the elements and values of a database. Their specification is far more technically oriented then business oriented. Some triggers implement business rules, but many implement and enforce data integrity and data consistency. Applications builders who are using a trigger built by another developer might have difficult deducing the business rules implemented by the trigger by looking at trigger code. Business analyst, the individuals who should be responsible for business rules specification, frequently find the triggers hard to learn.

Brief Summary Text (16):
More recently, object-oriented business rules technologies have evolved which allow rules to route work through the tasks of a business process, where reasoning can be applied to complex decision-making, and where knowledge systems can perform operator assistance.

Brief Summary Text (17):
Object-oriented business rules technologies base rule processing on an application's object model or component model. Some products based on these technologies use inferencing techniques on an application's object model to create, delete, and manipulate variables and objects and to determine their values. Other products utilize a technique which always fire a rule before or after an object method. Both of these techniques are very programmer intensive, as they are built right into the objects themselves.

Brief Summary Text (18):
Business rules are different from Business Objects. Business Objects represent business entities like customers, products, and orders. They encapsulate the data and behavior needed to perform business functions. Business rules implement the policies and practices of an organization. They control the ways that Business Objects perform business functions. An example will demonstrate the differences. For order processing, an application uses customer, product, and order Business Objects. While orders are processed, values for customer credit limits and inventory restock levels come into play. An organization may embed these values within the Business Object. But using business rules to determine these values would be more effective because business rules separate business policies from application processing. The business policies can be easily changed to reflect dynamic business changes. A credit limit may depend on factors including current receivables, credit history, time of year, product demand, product life cycle, creditor's business stability, and so on. Most Business Objects would have a hard-coded value for credit limit. Business rules can "reason" a value based on complex and timely criteria. Business rules can make a business more adaptable to business changes, more responsive to its customers, and more responsible to its stockholders.

Brief Summary Text (19):
Object-oriented technologies give business rules tremendous power and flexibility. Rules may be applied to a broad range of application resources. Individual rules may specify multiple conditions, and developers may specify complex relationships among the conditions. Systems of rules may be executed to "reason" a complex decision. And the technologies use backward chaining and/or forward chaining to iteratively and recursively evaluate rules.

Brief Summary Text (20):
The disadvantage of object-oriented business rules technologies is the traditional trade-off of power and flexibility--complexity. Use of these technologies requires skills in object-oriented design and object-oriented development. Specific products are built on proprietary and idiosyncratic object models and use proprietary languages for business rule specification.

Brief Summary Text (21):
Consequently, it would be desirable to provide a procedure for externalizing business decisions into business rules which are described and manipulated by business experts instead of programmers.

Brief Summary Text (23):
This invention relates to a method and apparatus for identifying decision points in Business Objects and classifying business rules that are applicable to the decision

points. Business Objects are created which are decorated with business rules, either manually or programmatically. Control Points are used to represent the named decision points or triggers within the behavior of the business objects. The Control Points are visually exposed to business analysts to allow the examination of the business rules attached to them. The business analysts are permitted to manually attach or detach business rules associated with the Control Points. A system can programmatically decide which rules to attach to a Control Point based on execution context.

Drawing Description Text (2):
FIG. 1 is an object interaction diagram showing the manual attachment of business rules to a Business Object;

Drawing Description Text (3):
FIG. 2 is an object interaction diagram showing the attachment of business rules to a Business Object using an object factory;

Drawing Description Text (4):
FIG. 3 is an object interaction diagram showing the attachment of business rules to a Business Object from a rules repository;

Detailed Description Text (2):
This invention provides a technique for identifying decision points in a Business Object and classifying business rules that are applicable to those decision points. Control Points are used to identify the decision points or triggers within the behavior of the Business Objects. Triggers are encountered in the semantic interface of the Business Object and trigger a named Control Point. Through the named Control Points, triggers are surfaced to authorized business analysts. This permits the business domain expert to examine the rules attached to the various Control Points, and at their discretion, manually attach or detach business rules. The availability of the Control Points allows the business domain experts to reuse a decision point in different business operations of the Business Object. In addition, the Control Points provide a place to cache the business rules across multiple trigger points in the Business Object. A place is provided within the Business Object, via the Control Points, to hang requirements of the decision point with respect to the acceptability of business rules (e.g., parameters provided and result expected).

Detailed Description Text (3):
Referring now to FIG. 1, there is shown an object interaction diagram 10 for manually attaching business rules 12 to a Business Object 14. After the Business Object 14 has been created, it can be decorated with business rules 12, either manually (as shown in FIG. 1) or programmatically (as shown in FIGS. 2 and 3). When a business rule 12 is attached manually, the user performing the attachment function visually associates the business rule 12 with one of the exposed Control Points in the Business Object 14. The user performing the task picks up the business rule 16 and drags and drops it onto the identified Control Point 18 in the Business Object 14. This triggers a validation step 20 whereby the Business Object hosting the Control Point describes the parameters it is prepared to provide and the result that is expected. The business rule 12 looks at this descriptive information and decides whether it is a likely fit for that Control Point. If the validation succeeds, the business rule 12 is attached to the Control point 22. Once the business rules 12 are associated with a specific Control Point in the Business Object 14, determining which rules to fire during execution of a business transaction is relatively straight forward. When a decision point or trigger is encountered in the programmed logic of the Business Object 14, the object looks to see if there are any business rules attached to the associated Control Point. Only those business rules attached to that Control Point are fired 24 at that time and the appropriate rules are executed 26.

Detailed Description Text (4):
Turning now to FIG. 2, there is shown an object interaction diagram 30 for programmatically attaching business rules 12 to a Control Point in a Business Object 14. A Factory Object 32 can perform the attachment 36 of business rules 12 after the Business Object 14 has been created 34. The Factory Object 32 has access to collections of business rules sorted and identified by their associated Control Points. Because the business rules are prevalidated when they are placed into the rule collections, no validation of parameter and result compatibility is needed prior to attaching the rule 38 to the Control Point. The Control Points are used by the Factory Object 32 to tell the Business Object 14 where to attach 38 the various

business rules. Once the business rules 12 are associated with specific Control Points in the Business Object 14, determining which rules to fire during execution of a business transaction is relatively straight forward. When a trigger is encountered in the programmed logic of the Business Object 14, the object looks to see if there are any business rules attached to the associated Control Point. Only those business rules attached to that Control Point are fired 40 at that time and the appropriate rules are executed 42.

Detailed Description Text (5):
Referring now to FIG. 3, there is shown an object interaction diagram 50 for periodically attaching business rules from rule repositories 51 to pre-defined Control Points in a Business Object 14 in order to reduce the rule identification processing that must take place during the execution of business logic. When the Business Object 14 is executed, one or more triggers may be encountered. These triggers which are associated with Control Points, represent the predefined triggers for firing applicable business rules. In addition to describing the decision points within a Business Object, the name space of Control Points is also used to categorize business rules. Knowing the identity of a Control Point triggered by a Business Object 14 makes it possible to search through the various sets of business rules to find the subset that is applicable to the triggered Control Point. The resulting set of applicable business rules can be constrained by the various jurisdictions (e.g., A range of authority. Jurisdiction objects are used to partition the space of business rules, such that when a trigger point is encountered, interested jurisdictions contribute the rules to be fired), the business activity taking place, and/or the agent performing the activity. The rule retrieval process can be optimized to only obtain rules when a Control Point is triggered, or it can be optimized to obtain rules for all Control Points in advance of their being triggered with either optimization. As each Control Point is encountered, the set of rules are cached when they are attached 62 to their respective Control Points, making the search step unnecessary when a Control Point is encountered during the normal execution of business logic. When the set of asserting jurisdictions changes, the business activity changes, or the performing agent changes 52, the cache within the Control Point can be invalidated and a new set of applicable business rules can be identified and attached 54. The Control Points 56 for the new set of applicable business rules are obtained from the business object 14, and a search is performed 58 in rule repositories 51 to identify the applicable business rules and return them 60 to the business object 14. The process of periodically acquiring the set of applicable rules and attaching them to Control Points ensures that the right set of rules is always known to the Business Object 14. Thus, when a Control Point is encountered, the set of rules to fire has already been identified and saved through attachment, and firing can immediately proceed.

CLAIMS:

1. A method, implemented in a computer system, for identifying decision points in an object-oriented object, comprising the steps of:

creating said object having a plurality of decision points in said computer system;

identifying and naming points of attachment for business rules in the operation of said object for each of said plurality of decision points;

associating by name a selected one of said plurality of decision points to at least one trigger point in said object;

assigning selected business rules to said named points of attachment for business rules in said object to said named associated decision point;

executing said object in said computer system until said at least one trigger point is detected in said object attached to said named associated decision point; and

displaying to a user in said computer the points of attachment for business rules in said object and the selected rules which have been associated with said named decision point.

2. The method of claim 1 wherein the step of associating further comprises:

assigning by name a selected one of said plurality of decision points to a different

trigger point in said object to allow reuse of at least one of said plurality of decision points to different business operations of said object; and

assigning acceptance criteria for the selected business rules to said associated decision point for said trigger point.

3. The method of claim 1 wherein the step of executing said object further comprises:

identifying a new set of business rules at a selected one of said points of attachment for business rules in said object;

verifying the acceptability of said identified business rules to said selected one of said points of attachment for business rules using rule acceptance criteria;

attaching said new set of business rules to said object to affect the operation of said object; and

executing said new set of business rules for said selected one of said points of attachment for business rules for said object.

4. The method of claim 1, wherein the step of executing said object further comprises:

examining the selected business rules attached to points of attachment for business rules in said object by said user;

identifying a set of replacement rules at a selected one of said points of attachment for business rules in said object;

manually attaching said set of replacement rules to said object by said user to affect the operation of said object; and

deleting at least one of said selected rules from a different one of said points of attachment for business rules by said user.

5. An apparatus for identifying decision points in an object-oriented object, comprising:

means for creating said object having a plurality of decision points in a computer system;

means for identifying and naming points of attachment for business rules in the operation of said object for each of said plurality of decision points;

means for associating by name a selected one of said plurality of decision points to at least one trigger point in said object;

means for assigning selected business rules to said named points of attachment for business rules in said object to said named associated decision point;

means for executing said object in said computer system until said at least one trigger point is detected in said object and executing rules attached to said named associated decision point; and

means for displaying to a user in said computer the points of attachment for business rules in said object and the selected rules which have been associated with said named decision point.

6. The apparatus of claim 5 wherein said means for associating further comprises:

means for assigning by name a selected one of said plurality of decision points to a different trigger point in said object to allow reuse of at least one of said decision points to different business operations of said object; and

means for assigning acceptance criteria for selected business rules to said associated decision point for said trigger point.

7. The apparatus of claim 5, wherein the means for executing said object further

comprises:

means for identifying a new set of business rules at a selected one of said points of attachment for business rules in said object;

means for verifying the acceptability of said identified business rules to said selected one of said points of attachment for business rules using rule acceptance criteria;

means for attaching said new set of business rules to said object to affect the operation of said object; and

means for executing said new set of business rules for said selected one of said points of attachment for business rules for said object.

8. The apparatus of claim 5 wherein the means for executing said object further comprises:

means for examining the selected business rules attached to points of attachment for business rules in said object by said user;

means for identifying a set of replacement rules at a selected one of said points of attachment for business rules in said object;

means for manually attaching said set of replacement rules to said object by said user to affect the operation of said object; and

means for deleting at least one of said selected rules from a different one of said points of attachment for business rules by said user.

9. A computer program product having a computer readable medium having computer program logic recorded thereon for identifying decision points in an object-oriented object, comprising:

computer readable means for creating said object having a plurality of decision points in a computer system;

computer readable means for identifying and naming points of attachment for business rules in the operation of said object for each of said plurality of decision points;

computer readable means for associating by name a selected one of said plurality of decision points to at least one trigger point in said object;

computer readable means for assigning selected business rules to said named points of attachment for business rules in said object to said named associated decision point;

computer readable means for executing said object in said computer system until said at least one trigger point is detected in said object and executing rules attached to said named associated decision point; and

computer readable means for displaying to a user in said computer the points of attachment for business rules in said object and the selected rules which have been associated with said named decision point.

10. A computer program product of claim 9 wherein said computer readable means for associating further comprises:

computer readable means for assigning by name a selected one of said plurality of decision points to a different trigger point in said object to allow reuse of at least one of said decision points to different business operations of said object; and

computer readable means for assigning acceptance criteria for selected business rules to said associated decision point for said trigger point.

11. A computer program product of claim 9, wherein said computer readable means for executing said object further comprises:

computer readable means for identifying a new set of business rules at a selected one of said points of attachment for business rules in said object;

computer readable means for verifying the acceptability of said identified business rules to said selected one of said points of attachment for business rules using said rule acceptance criteria;

computer readable means for attaching said new set of business rules to said object to affect the operation of said object; and

computer readable means for executing said new set of business rules for said selected one of said points of attachment for business rules for said object.

12. A computer program product of claim 9 wherein said computer readable means for executing said object further comprises:

computer readable means for examining the selected business rules attached to points of attachment for business rules in said object by said user;

computer readable means for identifying a set of replacement rules at a selected one of said points of attachment for business rules in said object;

computer readable means for manually attaching said set of replacement rules to said object by said user to affect the operation of said object; and

computer readable means for deleting at least one of said selected rules from a different one of said points of attachment for business rules by said user.

L2: Entry 35 of 42 File: USPT May 14, 2002

DOCUMENT-IDENTIFIER: US 6389588 B1
TITLE: Method and system of business rule extraction from existing applications for integration into new applications

Abstract Text (1):
A method of extracting and transforming a business rule which is a self contained section of legacy code focused on the computation of specific business policy includes identifying the business rule. Thereafter, the business rule code is located in the existing program and extracted in human readable code form. New code is generated for a new application for containing the business rule therein, and the new code is integrated into the new application. A system for extracting and transforming such business rules from existing programs such as legacy applications to a new application includes various components for achieving the various noted steps.

Brief Summary Text (7):
For purposes of the invention, a "business rule" is a self contained section of legacy code that is focused on the computation of a specific business policy expressed by a computation of value of a single or group of variables or, alternatively, by establishing an outcome of a specific business decision. Typical examples of such business rules include calculation of a bond's yield to maturity, determination of employment status, approval of loan eligibility or fulfillment of graduation requirements. Business rules can be described or defined at different levels, with a business rule often being composed of multiple business rules where it may become desirable to extract a higher level business rule, or one of the lower level business rules of a multiple number of business rules making up the higher level business rule. For example, a higher level business rule may be a subroutine that a banking system uses to qualify a user and determine a customer's mortgage. Such a higher level business rule can include lower level business rules such as the analysis done to determine whether: (1) the customer qualifies; (2) the availability of second mortgages; (3) determining whether based on customer data the customer merits a premium account; and/or (4) what is the best mortgage for the customer based on customer data. Thus, based on this description, it would be readily apparent to those of ordinary skill in the art what is intended by the term "business rule" as used herein.

Brief Summary Text (8):
It is well known that such business rules in legacy applications can be scattered throughout the entire program, and are most likely mixed in with other pieces of program logic. Further, under a typical legacy system, the mission critical business logic, or "business rules" therein are neither well documented nor easy to understand. The process of rediscovery is typically painful and costly, and once the legacy application, e.g., COBOL-based, "business rule" is isolated, the conversion of monolithic procedural code into a modern program language such as Java and the like must be dealt with in a manner which results in understandable hierarchies.

Brief Summary Text (9):
Thus, in implementing "business rules" from legacy applications in modern programs and platforms, the business rule must first be identified. One example of such an identification involves the end user of a COBOL program. Such an end user knows that the COBOL program does a value calculation that kicks off a sales process. Specifically, the program determines how often customers should be called based on the specific set of product lines they own. No one knows the basis on which the calculation is made. This may be identified as a "sales call rule" and is thus an ideal candidate for extraction as its own modular object. In accordance with the

necessary steps to be taken to find and extract such a business rule, the rule in the legacy source code must be located and extracted as a self contained routine. If the legacy code is well structured, the extraction of a business rule may be as simple as "cut and paste". However, most legacy code presents a number of challenges because such a rule can be scattered, for example, in a program of approximately 4,000 lines of code or more.

Brief Summary Text (15):
In recognition that a business rule can be made up of multiple separate business rules, for example, as in the case where a mortgage calculation will vary from state to state, such a business rule may be made up of: (1) a business rule outlining how the mortgage calculation is generally performed; and (2) a second business rule adjusting the mortgage calculation for a specific state. In such a case, a blocking technique can be used which is made up of backtracking through the existing code using the graphical editor to extract only the portions of the existing code making up the portions of the business rule desired to be extracted. Thus, by specifying blocking points, blocking allows the business rule extraction to stop the backtracking of the calculation at certain predetermined points selected by the user. Such blocking can be established as being of two types. The first type is known as generic blocking and stops the business rule extraction process at some given statement types, for example, at any "input/output" statement. A second type of blocking known as specific blocking stops the business rule extraction process at some statements which are predetermined and specified by the user.

Detailed Description Text (2):
Having briefly described the invention, the following detailed discussion presents a specific implementation of the invention, for example, as in the case of extracting a business rule or many business rules from an existing legacy program, for example, a legacy program written in COBOL.

Detailed Description Text (6):
Thus, in general implementation, the method and system of the invention is implemented in accordance with FIG. 3 wherein the original legacy application 17 such as one written in COBOL, PLI, or C, is analyzed to identify, locate or extract the business rules at a step 19. The business rules 21 are then transformed and integrated at a step 23 into a target application, for example, an application written in Java, C++, other languages, including even COBOL, generally referred to by the number 25, which is deployed in a computer system 27 which can be a standalone system or a distributed environment. Generally, as previously discussed, the process of business rule extraction includes the steps of identifying the existence of the business rule, locating the business rule in the legacy code, extracting the business rule as an independent entity, transforming the business rule into the new platform language, and integrating the business rule into the new system.

Detailed Description Text (12):
With respect to the business-rule integration, modern Java environments provide a great deal of flexibility for adopting the newly re-deployed business rule to work with a variety of distribution mechanisms. Once the logic is translated, a decision is made on how to integrate the resulting Java business rule into the new system. There are numerous ways in which the extracted business rules could be used, for example, the calculation which previously existed as a part of the underlying COBOL business logic, can now be included in a HTML page and distributed on request. In this matter, one can create a Java applet, in a manner which is conventional and well known to those of ordinary skill in the art once the invention as disclosed herein is known.

Detailed Description Text (15):
Thereafter, the business rule is transformed in a process of translating, as described previously, an extracted business rule from its original language, e.g., COBOL, PL/I, and C, into a new language of choice, e.g., Java, C++, Visual Basic and/or COBOL. The translated business rule is syntactically and semantically correct as further described herein, and accurately captures the business behavior of the legacy application. The business rules can then be further enhanced in the target integrated development environment to reflect new business requirements. Finally, transformed business rules are integrated into new applications. Component object models, like CORBA, COM/DCOM and Enterprise Java Beans are best suited for wrapping business rules to be plugged into modern technical infrastructure.

Detailed Description Text (18):
Thereafter, program analysis is conducted at a step 33 through the use of a graphical editor, such as the previously discussed Hyperview.TM. graphical editor or other like graphical editor, to allow the user to analyze the programs and find relevant business rules. Thus, using such a graphical editor, the user can see relationships between various elements of the program, for instance, the user can see all instances of a variable usage, the data structure used in a "Write" or a "Read" statement, or all "IF" statements that use a particular variable. In addition, using the graphical editor, the user can see a diagram of calls between program paragraphs. The user can also see a tree view of the divisions, sections, paragraphs and statements in a legacy application. e.g., in a COBOL program, and can see various execution paths in the program.

WEST

Generate Collection Print

L2: Entry 24 of 42 File: PGPB Aug 29, 2002

DOCUMENT-IDENTIFIER: US 20020120917 A1
TITLE: Business rules user inerface for development of adaptable enterprise applications

Abstract Paragraph (1):
Methods and apparatus, including computer program products, for interacting with a user to define business rules in a declarative manner. The invention operates to display a rule set as an editable list of conditions and an editable list of actions, the conditions and actions being linked to each other by the combination of an editable list of if-values and an editable list of then-values, wherein if-values and then-values are explicitly linked to each other, conditions and if-values are explicitly linked to each other, and then-values and actions are explicitly linked to each other in the displayed lists.

Cross Reference to Related Applications Paragraph (1):
[0001] This application claims priority on the basis of commonly-owned U.S. Patent Application No. 60/250,869 for Business Rules User Interface Elements For Development Of Adaptable Enterprise Applications, filed Dec. 1, 2000, the disclosure of which is incorporated here by reference in its entirety.

Summary of Invention Paragraph (3):
[0003] An enterprise application is a software program used by business people to increase productivity through automated data processing. Enterprise applications put into action a set of business requirements, expressed using natural language and "business speak". For the purposes of better defining the system, business requirements can be broken down into a set of interrelated business rules. A business rule, as defined by the GUIDE Business Rules Project, is a statement that defines or constrains some aspect of the business. A business rule is intended to assert business structure, or to control or influence the behavior of the business. A business rule should be atomic so that it cannot be broken down further without losing meaning.

Summary of Invention Paragraph (4):
[0004] Traditional implementations of business rules typically involve hard-coding them as procedural (flow-chartable) logic using programming languages such as Java, C++, and COBOL. Such business logic generally implements the business requirements of the enterprise application, providing the instructions necessary to store, retrieve, create, and manipulate data as well as validation constraints on such transactions. Implementing business logic using such languages requires highly trained software engineers and is relatively expensive and time-consuming. In addition, procedural programming languages do not support inferencing, which is a key feature of a robust rule-based system. They also make it prohibitively difficult to identify and resolve logic errors such as ambiguity and incompleteness among interrelated business rules.

Summary of Invention Paragraph (5):
[0005] An alternative approach to business rules automation is through expensive and difficult-to-use expert (rule-based) systems. This entails the conversion of the business rules into a formalized syntax that more closely represents the declarative nature of the business rules, generally requiring the services of highly trained knowledge engineers. The formalized rules are then processed through an inference engine, which itself requires tremendous programming effort to integrate with existing enterprise systems. Although some such rule-based systems provide the ability to easily modify the business rules, they fail to support certain business rule types such as constraints. And like procedural implementations, they provide no

support for resolving logic errors.

Summary of Invention Paragraph (7):
[0007] However, adapting component business logic requires manipulation of programmed code, which is difficult and, in any event, generally not allowed under the component license agreement. Although component behaviors may be influenced by parameterization, they are nevertheless limited to the problem domain contemplated by the component developer. On the other hand, declarative business rules not hard-coded into a component can be easily adapted to accommodate changing business requirements. A method for the automation of such rules is the inference engine. However, inference engines have been difficult to use within server-side component-based systems, with integration requiring coding to the proprietary API of the inference engine, and processing instructions represented in the proprietary syntax or programming language of each particular inference engine.

Summary of Invention Paragraph (10):
[0009] The invention can be implemented to realize one or more of the following advantages. A platform can be implemented in accordance with the invention that reconciles component and business rules technologies, combining the reusability features of component technologies with the adaptability features of business rules to create a powerful unified platform. One particular implementation of the platform integrates standards such as the Unified Modeling Language (UML), Enterprise JavaBeans (EJB), and Extensible Markup Language (XML) with business rule technologies. It enables non-technical business experts to play an active and central role in the development process of highly adaptable business applications. It offers a highly effective development methodology, an integrated set of standards-based tools, and a robust, scalable deployment platform. It provides an optimal environment for diverse enterprise applications.

Summary of Invention Paragraph (12):
[0011] The platform systematically separates business rules from procedural business process logic and thereby improves code quality and reduces development and maintenance costs. This makes rules technology accessible to mainstream developers and business experts. The platform enables non-technical personnel to develop, test, deploy and update sophisticated business rules declaratively, with no need for procedural programming, allowing for complex dynamically adaptable application behavior. These benefits are amplified for applications whose rules are volatile or subject to frequent changes, as well as applications impossible or prohibitively difficult to implement procedurally due to their logical complexity.

Summary of Invention Paragraph (14):
[0013] Enterprise software projects tend to suffer from schedule and budget overruns. Some are a total failure in delivering the wrong solution that does not meet the business needs. The primary reason for such shortcomings of the traditional approaches is the propensity to initiate coding before the business problem and the appropriate solution are fully understood. Unfortunately, from a management point of view, progress tends to be measured by tangible deliverables. This early development of business rules, sometimes hand-in-hand with rapid prototyping of presentation elements, offers such tangible deliverables to help satisfy the progress-hungry business managers, while minimizing the high risks associated with the late discovery of logic errors.

Summary of Invention Paragraph (16):
[0015] The early stage focus on development and clarification of business rules has all the benefits of rapid-prototyping without the shortcomings. Not only is the business logic completely fleshed out prior to programming, but also the resultant rules are not a throwaway. In fact, they become the cornerstone of the rest of the development lifecycle.

Summary of Invention Paragraph (17):
[0016] Enterprise Java business components tend to be thinner than alternative architectures such as CORBA and COM. This is because necessary middle-tier services such as security, concurrency control, transaction, and lifecycle management are not coded directly into the component, but are delegated to be handled by the component container. Nonetheless, Enterprise Java components still contain process or transactional logic and embedded business rules. The further extraction of business rules from such components has the added benefit of reducing their complexity even more. Consequently, the remaining logic of such components can be generated automatically from design models, minimizing the need for programming. Business

rules tend to be the most volatile part of a business application. It is thus advantageous to maintain them using a highly adaptable environment. It is also advantageous to make them accessible to non-programmer business experts who can implement the changes directly. Traditional programming languages do not have these features and advantages.

Brief Description of Drawings Paragraph (7):
[0023] FIG. 5 shows a business object model displayed graphically by a user interface.

Detail Description Paragraph (5):
[0028] The middle tier of the development platform 160 implements the business logic of an application. In prior systems, business logic was implemented procedurally, requiring programmers to build and maintain application code. In contrast, the platform 160 implements a visually declarative approach. With the platform, business logic can be subdivided into two parts: business rules and process logic.

Detail Description Paragraph (6):
[0029] Business rules, which often make up the bulk of business logic, are built and maintained in the visually declarative environment of a rules IDE 180. This allows non-technical developers, such as business experts and analysts, to create and maintain enterprise business rules as reusable, adaptable components, which will be referred to as rulepacks. Rulepacks are declarative components that encapsulate a set of business rules automating the knowledge of a particular context of the business process. Rulepacks are made up of one or more rulesheets. Each rulesheet can contain one or more rules that apply within the same particular scope. Rulepacks are built and maintained by the IDE 180.

Detail Description Paragraph (7):
[0030] The IDE includes a vocabulary 181, which represents the business entities, their attributes, and their associations (relationships) in the form of a tree view. The vocabulary can be created within the IDE or imported from a UML business object model (also known as a class diagram) 174, or generated from business objects, from a relational database, or from an XML schema. The vocabulary tree view serves as an optimal drag-and-drop source for easy creation of rulesheets (see, e.g., FIG. 6). The IDE also includes a rulepack and rulesheet editor 182, which is a visual environment designed to be used by non-technical business domain experts to build and maintain rulepacks and their rulesheets. A rulesheet is a spreadsheet-like construct for intuitive development of logically correct sets of rules. The visual environment created by the rulepack editor 182 is illustrated in FIGS. 6-18.

Detail Description Paragraph (8):
[0031] Rulepacks, in the present implementation, are implemented as XML documents expressed in a rules markup language created for that purpose. The rules markup language defines an XML rules interchange format similar to that of the Business Rules Markup Language (BRML), which was defined by the CommonRules Java library, available from International Business Machines Corporation of Armonk, N.Y. The syntax of the rules markup language is shown in Table 1 at the end of this specification represented in an Extended BNF (Backus Normal Form).

Detail Description Paragraph (11):
[0034] A developer will use the UML modeler and code editor 170 to build and maintain a use case model, a business object model (a UML class diagram), a component model, and various other UML models necessary to implement a complete enterprise application. For these purposes, the modeler and editor 170 includes a use case modeler 172, a business object modeler 174, and a component modeler 176, and other UML modelers. The modeler and editor 170 also includes an IDE (Integrated Development Environment) 178 that supports a Java development kit (JDK), optionally extended with a Java math library to support business domain calculations.

Detail Description Paragraph (12):
[0035] The use case modeler 172 is used to build and maintain business requirements visually in the form of UML-standard use cases. The business object modeler 174 is used to build and maintain an enterprise-level object model of all data elements. The enterprise-level model represents all data in the enterprise. It can be created from scratch or derived from existing enterprise databases. The objects of the enterprise-level object model also contain business functions. Business rules that are non-declarative in nature or involve an algorithm or complex mathematical calculation are captured as functions. The procedural component modeler 176 is used

to build and maintain those procedural business components that use rulepacks.

Detail Description Paragraph (14):
[0037] As shown in FIG. 1, the middle tier of an application includes a web server 120, a Java application server 130, and a business intelligence server 140. Any web server supporting JSP (Java Server Pages) and any Java application server can be used, such as a J2EE (Java 2 Enterprise Edition) compliant application server. In alternative implementations, the middle tier can be implemented using Microsoft.RTM. distributed Component Object Model (COM)-based technologies, including Active Server Pages (ASPs) and Microsoft.RTM. Transaction Server (MTS). The middle tier can also be implemented on a Microsoft .NET platform.

Detail Description Paragraph (16):
[0039] A rule component 142 is preloaded with the rules in a rulepack for optimal performance. The rule components 142 can interact with various types of business components (not just Enterprise JavaBeans components) using standardized messaging (e.g., XML messaging). The business intelligence server 140 provides a central integration hub that can interact with diverse application components 132, such as Microsoft.RTM. COM (Component Object Model) components, CORBA (Common Object Request Broker Architecture) components, EJB components, and Java components. The business intelligence server 140 can turn any Java application server into an application integrator, acting as a control center for an enterprise information system.

Detail Description Paragraph (18):
[0041] A simple application development methodology that takes advantage of the features of the application development platform 100 will now be described. Unlike traditional techniques that focus on a combination of data and process, or on objects that encapsulate both data and process, the methodology places business rules in the center of the development process.

Detail Description Paragraph (19):
[0042] Business rule analysis, as facilitated by the platform 100, accelerates the software development lifecycle by reducing unstructured requirements verbiage into concrete statements that are easily verified by business users. In addition, the methodology enables analysts and developers to identify, capture and test business rules from the very inception of a project. In addition, the methodology guarantees that UML-conforming documentation (such as use case models) will be created as an inherent byproduct of the software development process. Furthermore, the methodology ensures that the documents remain in synchronization with the application logic, because the business rules that the documents represent are literally part of the application.

Detail Description Paragraph (20):
[0043] By allowing developers to begin building and testing business rules at project inception, the methodology ensures healthy design disciplines, such as providing tangible design deliverables in a project's early phases. Moreover, business rules developed early in the development process are directly incorporated into the final application, resulting in cost savings when compared with traditional "throw away" prototyping techniques.

Detail Description Paragraph (22):
[0045] A business object model can be created from scratch or derived from an existing database (step 330). The business object modeler 176 can capture the business object model as a UML class diagram. An example of such a diagram is diagram 504 shown in user interface window 502 in FIG. 5. The IDE 180 transforms this into a vocabulary tree view for easy drag-and-drop functionality onto the rulesheets 182. Such a vocabulary tree view is shown in pane 604 of FIG. 6. However, the vocabulary can also be created directly in the rule IDE 180, without the need for an object model. The vocabulary can optionally have a name, illustrated as "FIM" 622 in pane 604.

Detail Description Paragraph (24):
[0047] Next, business rules are developed and tested (step 340) using the vocabulary 181, the rulepack editor 182, and the tester 183. Business rules are captured as discrete rule sets (rulepacks). Business rules can be categorized into three types: constraints (rejecters), triggers (projectors), and derivations (producers). The rulepack editor 182 enables developers to compose all types of rules that operate on data structures defined in the business object model. Developers can drag-and-drop terms from a tree view of the vocabulary 181 onto a rulesheet in order to define the

conditions and actions of each rulepack.

Detail Description Paragraph (25):
[0048] FIG. 6 shows a rulesheet containing constraints. Constraint rules allow developers to specify business rules that constrain a business and therefore define the boundaries for its valid state. Constraint rules are generally constraints such as data validation and integrity rules. The vocabulary 181 (FIG. 1) of the business object model is displayed in an equivalent tree view in a pane 604 at the left of the rulesheet. To the right of the tree view, the possible conditions (upper left quadrant 606) and actions (lower left quadrant 608) are tabulated. Because the word "condition" has different meanings according to context, the expression in a row in the upper left quadrant will occasionally be referred to as a "condition term" for the sake of clarity. The order in which actions are listed in the actions quadrant 608 can optionally be used to determine the order in which the actions are executed. However, it is advantageous that this not be done, because doing so would introduce a procedural aspect to the definition of rules that is inconsistent with the declarative design of the user interface.

Detail Description Paragraph (30):
[0053] FIG. 7 shows in pane 704 a partially expanded vocabulary tree corresponding to the vocabulary tree shown in pane 604 of FIG. 6. All nodes of the tree at the root level, such as entity Account (node 706), are entities. The Account entity has been expanded by a user to show the attributes, such as the attribute number (node 708), of the Account entity. When an entity is expanded, its relationships with other entities, if any, are also shown. These relationships will have been defined, for example, by a UML business object model (also known as a class diagram), as described earlier.

Detail Description Paragraph (35):
[0058] FIG. 7 also shows a rulepack display 702 with multiple rulesheets displayed as tabs 720. The visible tab shows a rulesheet containing derivation rules (producers). Derivation rules allow developers to specify business rules that infer or calculate the value of derived fields. In this rulesheet the actions are statements that assign values to fields (i.e., they are assignment actions). The fields are identified in rows in the Actions pane, the values are in the corresponding then-value cells. In addition, this rulesheet also shows a pane 722 for entity shortcuts, called "Shortcuts", and a pane 724 for unconditional calculations, called "Rules (non-conditional)". The shortcuts link alias names to the object model. The non-conditional rules are business rules that fire without any conditions.

Detail Description Paragraph (45):
[0068] FIG. 13 shows how business rules can be tested immediately after they are coded, independent of other development activities. The left pane 1306 represents the input into a rules component 142. This can be developed using drag-and-drop from the vocabulary pane 1304 or directly from a database or a test suite repository. The right pane 1308 shows the output of the rules component, which is the contents of the left pane 1306 modified by the rule engine of the rules component. In FIG. 13, for example, the rule component calculated a value for the term dProfile for each of four different securities, which were provided as input in the input pane. The difference between the two panes can be highlighted by a tester subsystem to aid the developer in determining what rules fired. Additional features such as breakpoints and watch windows can be added to aid the user in stepping through the processing of the rulesheet.

Detail Description Paragraph (46):
[0069] While rules development and testing are in progress, developers can simultaneously code custom processes (or acquire pre-written components if available) and begin implementing the user interface. These activities produce feedback that can be used to refine requirements, UML models, and the business rules, resulting in an iterative development cycle. Development cycle iterations continue as necessary until the application is ready for deployment and then throughout its life as features are changed, upgraded or extended.

Detail Description Paragraph (57):
[0080] The invention has been described in terms of particular embodiments. Other embodiments are within the scope of the following claims. For example, the steps of the invention can be performed in a different order and still achieve desirable results. The invention can be implemented in other component architectures. For

example, the invention can be implemented using Microsoft APS (Active Server Pages) and COM (Component Object Model) or DCOM (Distributed Component Object Model) technologies or CORBA (Common Object Request Broker Architecture) rather than Java-based technologies, and programmed components of the invention can be programmed in a language other than Java, for example, C++.

CLAIMS:

1. A computer program product, tangibly stored on a computer-readable medium, for interacting with a user to define business rules in a declarative manner, comprising instructions operable to cause a computer coupled to a display device to: display a rule set as an editable list of conditions and an editable list of actions, the conditions and actions being linked to each other by the combination of an editable list of if-values and an editable list of then-values, wherein if-values and then-values are explicitly linked to each other, conditions and if-values are explicitly linked to each other, and then-values and actions are explicitly linked to each other in the displayed lists; and receive from a user inputs editing one or more of the editable lists.

38. A computer-implemented method for interacting with a user to define business rules in a declarative manner, the method comprising: displaying a rule set as an editable list of conditions and an editable list of actions, the conditions and actions being linked to each other by the combination of an editable list of if-values and an editable list of then-values, wherein if-values and then-values are explicitly linked to each other, conditions and if-values are explicitly linked to each other, and then-values and actions are explicitly linked to each other in the displayed lists; and receiving from a user inputs editing one or more of the editable lists.

WEST

Generate Collection Print

L2: Entry 1 of 42 File: PGPB Sep 4, 2003

DOCUMENT-IDENTIFIER: US 20030167456 A1
TITLE: Architecture for building scalable object oriented web database applications

Abstract Paragraph (1):
A method for modeling and rapidly building high performance object oriented database applications for web environments is disclosed. The modeling encompasses behavioral object modeling as well as structural data modeling according to a set of rules that yields a layered object model with no compromises on the database design, the application's functionality or code reusability and extensibility. A high level mechanism based on the Extensible Markup Language (XML) is used to declare the structure and behavior of modeled persistent objects that exhibit functionally complete object orientation and whose implementations are realized through packages of database stored procedures and associated structures. Code generators produce the necessary application and database code from the XML specification, enabling rapid development. The packages of stored procedures encapsulate all aspects of the database design and database programming, yielding performance, flexibility and future-proofing of the applications from changing requirements, database versions and database performance tuning. The generated code, in conjunction with a lightweight run time infrastructure, provides performance and development productivity features that are specifically geared for the stateless web environment in order to support scrolling of very large result sets from the database, to automatically detect conflicting changes from multiple concurrent users, and to automatically render the state of persistent objects in XML for personalization and data interchange. Additional performance features include high-concurrency caching of persistent objects with transactional semantics for ensuring transaction isolation among multiple threads of execution.

Summary of Invention Paragraph (28):
[0026] c. Poor functionality due to an artificial force-fit of a behavioral object model to a structural data model, or worse, a force-fit of a structural data model to a behavioral object model.

Summary of Invention Paragraph (39):
[0037] 4. The solutions conform to the premise that the implementation of all business rules belong in the middle tier, which is often referred to as the business logic layer. The consequence of this mode of thinking is poor performance coupled with a degraded user experience due to the artificial routing of processing that is better executed closest to its data coupled with the delayed response time to user error conditions.

Summary of Invention Paragraph (44):
[0041] 1. A high level XML vocabulary for declaratively specifying a model of the application's persistent state, behavior and relationships, referred to as its persistent component. The vocabulary also allows for embellishments to the object model to define a not-necessarily-equivalent data model and stored procedure definitions and implementations. The XML vocabulary is defined in an XML Document Type Definition (DTD) file or an equivalent XML Schema file. For a description of XML, DTD and XML Schema, please refer to the XML Standards Committee's web URL http://www.w3c.org.

Summary of Invention Paragraph (56):
[0053] 4. A methodology for modeling object oriented applications that have data persistence requirements. The key differentiator of the methodology is its use of both object and data modeling disciplines and the integration of persistent and non-persistent object models into a layered object model with well-defined rules of

interaction between the layers that are described below under Operation.

Summary of Invention Paragraph (90):
[0086] The methodology accomplishes these results by allowing a designer to think of persistence at a coarse-grained business object level rather than at the level of database access or query languages. A systematic approach is applied to arrive at a layered object model where application level objects maintain unidirectional relationships with persistent objects. WDO's support for fine-grained persistent objects facilitates the natural modeling of containment and aggregation relationships that result in reduced communication overhead with the database. Code reuse is accomplished at the application level by using WDO's support for inheritance and polymorphism.

Summary of Invention Paragraph (104):
[0100] A high degree of functionality and code reuse is possible through the support of full object oriented capabilities by WDO. Specifically, the following object modeling capabilities are provided, at a business object level:

Summary of Invention Paragraph (123):
[0119] At the object model level, it is sufficient to tag object attributes as interesting for the purposes of change detection. WDO automatically and efficiently manages the details of implementing optimistic concurrency control through detection of changes to the specified attributes.

Summary of Invention Paragraph (125):
[0121] WDO allows the object modeler to define list retrieval methods that manage list context. The associated design pattern permits a simple and efficient implementation in the database. This permits a web application to efficiently scroll through a potentially infinite sized list page by page such that successive page requests can be routed to different servers in a web farm for reasons of fault tolerance and load balancing.

Brief Description of Drawings Paragraph (4):
[0126] FIG. 3 illustrates the WDO modeling methodology. The processes of performing structural data modeling and behavioral object modeling are illustrated.

Brief Description of Drawings Paragraph (5):
[0127] An example of a layered object model that is arrived at using the WDO modeling methodology is illustrated in FIG. 4, which represents a trivial order entry application. The notation used in the object model is industry-standard UML (Universal Modeling Language). The figure shows how application level objects add value to persistent objects and reuse the persistent objects by inheriting from them and adding attributes and behavior that pertain to the middle tier. Also shown is the use of lightweight second-class objects that are managed by first class, or functionally complete, container objects. The first class persistent objects are shown with business-rule-level behavior.

Brief Description of Drawings Paragraph (6):
[0128] FIG. 5 illustrates the data model that corresponds to the example application illustrated in FIG. 4. The notation that is used is a widely used IEF (Information Engineering) notation also referred to as Crow's Feet. The data model illustrates how internal database structures are decoupled from what the application needs to see at a semantic level. In this case, certain internal primary key and foreign key columns are not visible at the object model level.

Detail Description Paragraph (4):
[0133] An object-oriented analyst and a database designer use the WDO modeling methodology to develop the behavioral and structural models for a given component respectively and arrive at an integrated object model. It is summarized below:

Detail Description Paragraph (5):
[0134] The analyst develops a behavioral object model of the web application component based on a use case analysis.

Detail Description Paragraph (7):
[0136] From the entity-relationship model, the database designer derives a preliminary persistent component object model. At this point, the model has classes with attributes and relationships but no useful methods.

Detail Description Paragraph (8):
[0137] The models are reconciled into a layered object model according to the following guidelines:

Detail Description Paragraph (13):
[0142] The following rules guide the process of reconciling the behavioral and initial persistent object model into a layered and integrated model as a path of least resistance towards a realizable and efficient application architecture.

Detail Description Paragraph (15):
[0144] The WDO vocabulary is expressed in a meta-language such as XML Schema or Document Type Definition. It specifies the arrangement of elements and attributes that constitute the vocabulary for defining the entities and attributes that constitute a persistent object model.

Detail Description Paragraph (71):
[0200] Next, the persistent object model in the lower layer of the layered model is represented in an XML file using the WDO XML vocabulary, and is embellished to add data model and other information so as to arrive at a self-contained definition from which all application and database code can be generated. This is accomplished using either a text editor or an XML editor.

Detail Description Paragraph (77):
[0206] All communication to the database consists of execution of database stored procedures. That is, no data manipulation SQL is ever executed from the middle tier. The implementations of the database stored procedures consist of SQL and procedural SQL code that actually operate on the database tables, both to access and manipulate data and to implement data-intensive business rules directly inside the database in a compact and efficient manner.

Detail Description Table CWU (1):
1 RULE REASON REMARKS Rule #1: A class defined in a Maintain unidirectional This rule can be relaxed WDO component cannot have a dependency between non-WDO somewhat when database stored has-a (containment), uses or is-a and WDO classes. procedures that are implemented (inheritance) relationship with a in an object oriented non-WDO class. programming language such as Java are used to interact with external systems for low- bandwidth operations. Rule #2: A WDO class can have Inheritance is supported. an is-a (inheritance) relationship with a WDO class. Rule #3: A non-WDO class can The memory management of a The capability is useful in a inherit from a first or second first class is completely handled layered object model. For class. If the former, its by its static member factory example, an order entry instantiation is possible only methods. It cannot be application's line item class with a first class factory method. relinquished to a derived non- may extend the attributes of A WDO class cannot inherit WDO class having additional an line item WDO class with from such a non-WDO class due attributes attributes, methods and to Rule #1. relationships that are relevant only to the run time environment. An application object that inherits from a first class can be cached and locked just like a first class. It may also provide client-side implementations for abstract first class methods. Rule #4: A second class cannot A second class is not much more A second class can be a base have useful methods or than a DBMS-resident structured class for a first class declared constructors defined on it. data type. within the same WDO component, but the reverse is not true. The DBMS implementation of the WDO classes can map both variants of a class to the same instance of the same underlying database structure (for example, a row of a table) if required. Rule #5: A first class cannot Also, see the reason for Rule #3. have constructors defined on it -- at least one set of static member methods is implicitly provided for instantiating an instance of the class. Rule #6: A first class can have As expected, an abstract WDO virtual methods defined on it. A class cannot be instantiated. virtual method may be an However, it does have available abstract method. the default static member methods for instantiation of application level derived classes. Rule #7: A non-WDO class can have a uses or is-a relationship with a WDO first or second- class. Rule #8: A WDO class may The restriction on classes that are contain another class. The allowed to be contained preserves contained class must be a WDO the unidirectional dependency class or a base Java class. relationship between the application and persistent object layers, and also simplifies the implementation.

CLAIMS:

1. A method for effectively modeling a scalable web database application for online

transaction processing that uses an object oriented programming language and a relational or object-relational database system, such that: a. neither functionality nor performance is compromised. b. all persistence related entities are isolated into a well defined layer that encapsulates database designs and operations on database tables. c. persistent objects are functionally complete objects complete with behavior, inheritance, polymorphism and containment over and above basic object oriented features such as state and identity. d. There is a partitioning of skill sets between database developers and object oriented application developers. by utilizing both object modeling and data modeling techniques and applying a set of rules for arriving at a layered object model where the objects in the layers interact according to well defined rules set forth in this patent application.